Restricted Scheduling Windows for Dynamic Fault-Tolerant Primary/Backup Approach-Based Scheduling on Embedded Systems

Petr Dobiáš, Emmanuel Casseau
Univ Rennes, Inria, CNRS, IRISA
Lannion, France
petr.dobias@irisa.fr

Oliver Sinnen
Department of Electrical and Computer Engineering
University of Auckland
Auckland, New Zealand

Abstract

This paper is aimed at studying fault-tolerant design of the real-time multi-processor systems and is in particular concerned with the dynamic mapping and scheduling of tasks on embedded systems. The effort is concentrated on scheduling strategy having reduced complexity and guaranteeing that, when a task is input into the system and accepted, then it is correctly executed prior to the task deadline. The chosen method makes use of the primary/backup approach and this paper describes its refinement based on reduction of windows within which the primary and the backup copies can be scheduled. The results show that the use of restricted scheduling windows reduces the algorithm complexity by up to 15%.

Index Terms

Dynamic scheduling, Fault-tolerance, Homogeneous faulty processors, Low complexity, Multiprocessors, Primary/backup approach

I. Introduction

At the present time, requirements for higher performance and lower energy consumption to meet the demands of more and more complex computations are increasing. To fulfil these goals, it is necessary to ensure system functionality and therefore design fault-tolerant systems. One of the possibilities to provide such systems is to make use of redundancy in space and time on multi-core platforms. Actually, they are less vulnerable when one processor is faulty because other processors can take over its scheduled tasks. Nevertheless, there exist several issues when conceiving a solution. It is mainly concerned with a trade-off between the parallel computing accounting for system performance and the spatial redundancy arranging for reliability, and with the dynamic mapping and scheduling control, which is in general an NP problem.

The main objective of this paper is to investigate how to dynamically schedule tasks onto homogeneous faulty processors with reduced complexity to execute the algorithm at run-time onto an energy efficient embedded system. The principal idea is to make use of the multi-processor system on chip and take advantage of its inherent properties. We developed a run-time algorithm based on the primary/backup (PB) approach reducing the scheduling window when searching for free slots for primary and backup copies and we evaluated its performances. The primary/backup approach is a commonly used technique to provide fault-tolerance, for example it can be adapted for satellites [1] or employed in virtualised clouds [2].

The rest of this paper is organised as follows. Section II sums up the related work. Our assumptions and our scheduling model are presented in Sect. III. Section IV describes and then analyses the experiments. Section V concludes the paper.

II. Related Work

Manimaran et al. [3] assumed that resources can be reclaimed not only after backup deallocation (if this technique is employed) but also when a copy finishes its execution earlier than scheduled.

Other authors study more elaborated techniques for overloading of task copies. Al-Omari et al. [4] introduced the *primary-backup overloading* allowing primary and backup copies of two different tasks to overlap each other on the same processor. It was found that this technique improves schedulability comparing to the basic version of backup-backup overloading but its reliability, measured by means of time to next fault (TTNF), can be diminished. Tsuchiya et al. [5] proposed the *active primary/backup approach* in order to schedule tasks with tight deadlines. This method, which is mainly interesting for real-time systems, enables primary and backup copies of the same task to overload each other on two different processors. However, there are system overheads owing to the overlap and the simultaneous execution of task copies.

Naedele [6] tried to improve the schedulability using the *decision deadline*, which authorised a task to be scheduled on probation if there is no free slots to place a new arriving task. In other words, when a new task T_i arrives on the system

This work was funded by the Reliasic project granted by the CominLabs Excellence Laboratory (ANR-10-LABX-07-01), French "Investissements d'Avenir" program.

and if it is not possible to immediately upon arrival determine schedules for primary and backup copies, this technique allows the algorithm to wait a certain amount of time, until the decision deadline, before it makes a decision whether the task T_i is rejected or not.

Several authors consider that the system can tolerate more than one fault at the same time. Manimaran et al. [3] proposed the algorithm, which divides the total number of processors into groups. There are as many groups as the maximum possible number of faults in the system all at once. Regarding that each task, i.e. its primary and backup copies, must be scheduled within one group, the more groups are considered, the less possibilities of available schedules and thus the higher rejection rate. Al-Omari et al. [4] improved the method allowing several faults in the system at one time by introducing the *dynamic grouping*. It means that groups are dynamically formed and dissolved depending on the task arrival and execution. It was shown that this method performs better but with system overheads than the one proposed by [3] or the conventional primary/backup approach.

Zheng et al. [7] noted that mapping and scheduling of backup copies have significant influence on the system schedulability. Consequently, they presented the algorithm, which computes the replication cost for each possible backup copy schedule and chooses the schedule having the lowest replication cost and, in case of tie, situated as soon as possible. In addition, they considered *boundary schedules* of backup copies defined as schedules having its start time and/or finish time at the same moment as boundaries of free slot or boundaries of overloadable backup copies. Such schedules enable the algorithm not to evaluate replication cost for all schedules on each processor and therefore lessen the complexity.

This section summarised the related work concerning the primary/backup approach. Most enhancements were proposed to improve the system schedulability or to diminish the rejection rate but without analysing and reducing the algorithm complexity.

III. Assumptions and Scheduling Strategy

The system consists of P homogeneous processors[1] accommodating aperiodic tasks, which are dynamically mapped and scheduled on the system without preemption. The system is equipped with a fault detection mechanism promptly informing about a fault occurrence. We consider that only one permanent or transient fault can occur at any instant of time.

Regarding our task model, each task has three attributes: arrival time a, computation time c and deadline d. The task window tw, defined as $d - a$, is a multiple α of c, i.e. $tw = \alpha \cdot c$.

The studied algorithm is based on the *primary/backup (PB) approach* [8], which is known for its minimal resources utilisation and high reliability. Its principal rule is that, when a task arrives, the system creates two identical copies: the *primary copy* PC and the *backup copy* BC. These copies are then scheduled on two different processors. Both copies must be scheduled within the task window and they are not permitted overlapping each other on different processors to prevent system overheads. The primary copy is scheduled as soon as possible and the backup one as late as possible in order to avoid idle processors just after the task arrival time and possible high processor load later.

In order to improve the schedulability and to minimise resources usage, we consider the backup deallocation and the backup overloading [8]. The former technique consists in deallocating the backup copy BC_i, if the primary copy PC_i was correctly executed. The latter one allows overloading of several backup copies, except ones having their primary copies scheduled on the same processor.

When implementing the backup overloading, we employed the method from [7], which maximises the backup overloading and makes use of boundary schedules. Thus, the algorithm computes replication cost to evaluate overlap percentage of arriving backup copy. If two possible schedules have the same replication cost, the one having the latter start time is chosen.

Algorithm 1 Studied algorithm

Input: Task T_i, Mapping and scheduling MS of already scheduled tasks
Output: Updated MS

```
1: if new task T_i arrives then
2:     for all (P) processors do
3:         Map and schedule PC_i (as soon as possible) within PC window
4:     end for
5:     for (P − 1) processors do
6:         Map and schedule BC_i (with minimum replication cost and as late as possible) within BC window
7:     end for
8:     if PC and BC schedules exist then
9:         Commit the task T_i
10:    else
11:        Reject the task T_i
12:    end if
13: end if
```

[1]In this paper for the sake of simplicity, a system with only homogeneous processors is considered. Nevertheless, the studied model can be easily extended to a system with heterogeneous processors, such as in [7].

The algorithm, presented in Algorithm 1, carries out the exhaustive search on all processors. Therefore, it finds the best solution, i.e. the one having the primary copy scheduled as soon as possible and the backup one maximising the backup overloading and placed as late as possible. On the one hand, this method is known to be the best for the PB approach regarding the rejection rate and processor load. On the other hand, the algorithm needs to test all boundary schedules within the scheduling window, which requires a non negligible number of comparisons. The aim of this paper is to show that the number of comparisons can be reduced.

A. Restricted Scheduling Windows

To lower the complexity when searching for a schedule, we define *scheduling windows* for primary and backup copies within which respective copy can be scheduled. Restricting the scheduling windows for both copies reduces the mutual scheduling interference and the complexity (measured by means of number of comparisons of schedules delimited by boundaries), and encourages to place the primary copies as soon as possible and the backup ones as late as possible, thus improving the system schedulability.

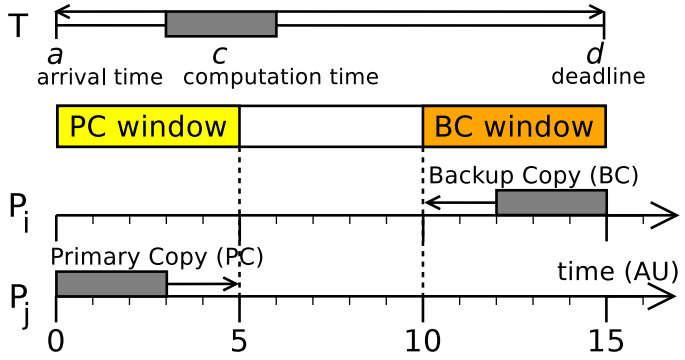


Figure 1: Primary/backup approach with restricted scheduling windows ($p = 1/3$)

The size of scheduling window is defined by a parameter p representing the percentage of task window. The primary scheduling window is thereby delimited by a and $a + p \cdot tw$ and the backup one by $d - p \cdot tw$ and d. Thus, for our algorithm the percentage is within $0 < p \leqslant 1$, while $p = 1.0$ for conventional algorithm.

Figure 1 depicts an example of restricted scheduling windows.

B. Evaluation of the Complexity

In this section, we theoretically focus on the value of the complexity when placing tasks using the restricted scheduling windows.

Inspired by [7], we define N_{ps} the number of all possible schedules where a copy can be placed. This number[2] is not easy to estimate in advance and that is why experimental results are essential to observe the trend. Regarding that, when searching for a backup copy schedule, there are more possibilities to examine (due to possible overloading), we denote $N_{ps}(PC)$ and $N_{ps}(BC)$ the number of all possibilities within the scheduling window when placing a primary copy or a backup copy, respectively. We remind the reader that we consider P processors and that a backup copy cannot be scheduled on the same processor as the primary copy. Thereby, the complexity $comp$ is defined as follows:

$$comp(PC) = P \cdot N_{ps}(PC) \cdot \max\left(\frac{1}{\alpha}; \min\left(1 - \frac{1}{\alpha}; p\right)\right) \tag{1}$$

$$comp(BC) = (P-1) \cdot N_{ps}(BC) \cdot \max\left(\frac{1}{\alpha}; \min\left(1 - \frac{1}{\alpha}; p\right)\right) \tag{2}$$

$$comp = comp(PC) + comp(BC) \tag{3}$$

An example is depicted in Fig. 2. It can be stated that, when the percentage of task window decreases, the complexity is reduced since there are less possible schedules to test. Nonetheless, we cannot theoretically evaluate system performances, such as rejection rate, and experiments are consequently necessary.

[2]To simplify, we consider that N_{ps} is uniformly distributed within the task window and has the same value on all processors.

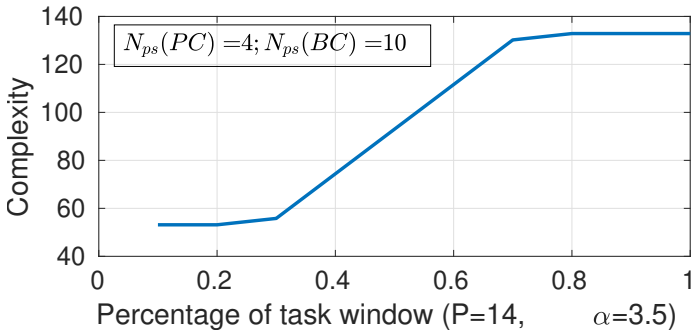


Figure 2: Example of theoretical complexity

IV. Experiments

A. Experiment Set-Up

Simulations were carried out to study the algorithm behaviour. Their parameters are summed up in Table I. For each simulation scenario, 100 simulations of 10 000 tasks were treated and the obtained values were averaged. We consider that no fault occurs and therefore all backup copies can be deallocated once their respective primary copies finish.

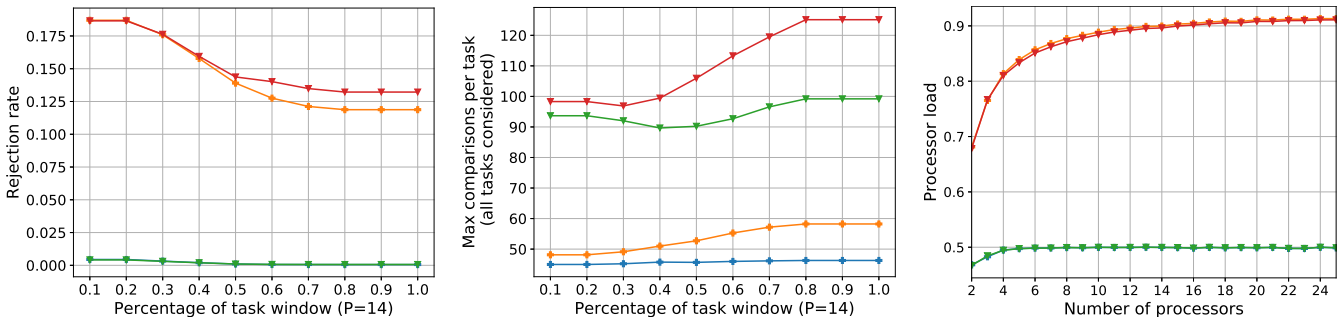
The *Targeted Processor Load* (TPL) is a parameter necessary to generate task arrival times. If $TPL = 1.0$, task attributes a and c are generated so that every processor is considered to be working all the time at 100%. The arrival time is defined using the Poisson distribution with parameter λ depending on computation time.

Table I: Simulation parameters

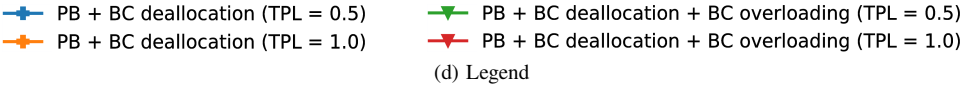
Parameter	Symbol	Distribution	Value(s)
Percentage of task window	p		0.1 – 1.0
Number of processors	P		2 – 25
Computation time	c	Uniform	1 – 20
Arrival time	a	Poisson	$\lambda = \frac{average\ c}{TPL \cdot P}$
Deadline	d	Uniform	$(a + 2c) - (a + 5c)$

In order to compare our refined algorithm to the basic one, the results of different percentage p are analysed, including $p = 1.0$, i.e. the case without the restricted windows as in the conventional primary/backup approach presented in [8].

B. Results



(a) Rejection rate (b) Maximum number of comparisons (c) Processor load ($p = 0.5$)



(d) Legend

Figure 3: Experimental results comparing primary/backup approaches based on the exhaustive search and on the boundary schedule technique with different percentage of task window and TPL

In this paper, most of the results are described for 14-processor system having the same input task set generated assuming $TPL = 0.5$ and $TPL = 1.0$, respectively. Actually, as depicted in Fig. 3c, it can be found that, when the number of processors

is greater than 12, observed metrics have already smooth trend and analogous results are obtained. The same conclusions were noted for example in [7].

Figure 3c, plotting the *processor load* for $p = 0.5$ as a function of the number of processors, illustrates that for each task set both methods, i.e. the PB approaches with and without overloading, overlap each other. Similar results are obtained for $p = 1.0$.

Figures 3a and 3b depict the studied metrics as a function of the percentage p of task window. These figures show that the represented curves remain constant from $p = 0.1$ to $p = 0.2$ and from $p = 0.8$ to $p = 1.0$, which is due to minimal considered ratio of computation times to task window in our experiment set-up ($2c \leqslant d \leqslant 5c$ and thus $c/d_{max} = 1/5$).

The *rejection rate*, represented in Fig. 3a, is the ratio of rejected tasks to all arriving tasks. First of all, it can be noticed that, if the percentage p drops below 0.5, the rejection rate climbs for all methods because of the rejection of tasks with hard time constraints. Then, we focus on p between 0.5 and 1.0 and it can be seen that, when $TPL = 0.5$, the rejection rate is the same for both methods, almost constant and close to 0%. For $TPL = 1.0$, the PB approach with deallocation slightly achieves better result than the PB approach with deallocation and overloading. When p decreases, the rejection rate increases because there are less possible schedules. The higher the TPL, the more noticeable this phenomenon. Nonetheless, in the worst case ($TPL = 1.0$), the difference between $p = 0.5$ and $p = 1.0$ does not exceed 2%.

The algorithm complexity is evaluated by *number of comparisons* accounting for the number of tested schedules delimited by boundaries. All tasks are taken into account, no matter whether they are finally accepted or rejected. This metric is essential since it is related to energy consumption and rapidity of mapping and scheduling. The evolution of the maximum value of this number is depicted in Fig. 3b. The figure points out that the PB approach with deallocation has lower number of comparisons, which is due to less possible schedules to test (no backup overloading). Moreover, if we focus on p between 0.5 and 1.0, the higher value of p, the higher number of comparisons.

To sum up performances of our proposed refinement of the PB algorithm, it can be stated that the number of comparisons arranging for the algorithm complexity presents the most notable change. In order to accept tasks having deadline between $2c$ and $5c$ and arriving on the system, we consider the restricted scheduling window fixed at $p = 0.5$. This value is thus a reasonable trade-off between the complexity and the rejection rate. In this case, the number of comparisons is reduced by 10% and 15% for PB approach with deallocation and PB approach with deallocation and overloading, respectively.

V. Conclusion

This paper is concerned with the dynamic mapping and scheduling of tasks on multi-core systems which are currently challenging problems of fault-tolerant design. We refined the algorithm based on the PB approach using restricted scheduling windows for primary and backup copies and aiming at reducing its complexity. It was shown that our algorithm has the complexity reduced by up to 15%, the processor utilisation stays the same and the rejection rate rises by at most 2% (compared to the conventional PB approach).

Taking into account the trade-off between the complexity and the rejection rate, the best results are obtained for the percentage of task window close to 0.5, i.e. when the first half of the task window is reserved for primary copies and the second half for backup ones.

References

[1] X. Zhu, J. Wang, J. Wang, and X. Qin, "Analysis and Design of Fault-Tolerant Scheduling for Real-Time Tasks on Earth-Observation Satellites," in *43rd International Conference on Parallel Processing*, 2014, pp. 491–500.

[2] X. Zhu, J. Wang, H. Guo, D. Zhu, L. T. Yang, and L. Liu, "Fault-Tolerant Scheduling for Real-Time Scientific Workflows with Elastic Resource Provisioning in Virtualized Clouds," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 12, 2016, pp. 3501–3517.

[3] G. Manimaran and C. S. R. Murthy, "A Fault-Tolerant Dynamic Scheduling Algorithm for Multiprocessor Real-Time Systems and its Analysis," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 11, 1998, pp. 1137–1152.

[4] R. Al-Omari, A. K. Somani, and G. Manimaran, "Efficient Overloading Techniques for Primary-Backup Scheduling in Real-Time Systems," in *Journal of Parallel and Distributed Computing*, vol. 64, no. 5, 2004, pp. 629–648. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731504000541

[5] T. Tsuchiya, Y. Kakuda, and T. Kikuno, "A New Fault-Tolerant Scheduling Technique for Real-Time Multiprocessor Systems," in *Proceedings Second International Workshop on Real-Time Computing Systems and Applications*, 1995, pp. 197–202.

[6] M. Naedele, "Fault-Tolerant Real-Time Scheduling under Execution Time Constraints," in *Sixth International Conference on Real-Time Computing Systems and Applications (RTCSA)*, 1999, pp. 392–395.

[7] Q. Zheng, B. Veeravalli, and C.-K. Tham, "On the Design of Fault-Tolerant Scheduling Strategies Using Primary-Backup Approach for Computational Grids with Low Replication Costs," in *IEEE Transactions on Computers*, vol. 58, no. 3, 2009, pp. 380–393.

[8] S. Ghosh, R. Melhem, and D. Mosse, "Fault-Tolerance Through Scheduling Of Aperiodic Tasks in Hard Real-Time Multiprocessor Systems," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, no. 3, 1997, pp. 272–284.